Merge Compose files

- How to merge multiple Compose files
- Merging rules
  - Additional information
- Example
- Limitations
- Reference information

Docker Compose lets you merge and override a set of Compose files together to create a composite Compose file.

By default, Compose reads two files, a `compose.yaml` and an optional `compose.override.yaml` file. By convention, the `compose.yaml` contains your base configuration. The override file can contain configuration overrides for existing services or entirely new services.

If a service is defined in both files, Compose merges the configurations using the rules described below and in the Compose Specification.

How to merge multiple Compose files

To use multiple override files, or an override file with a different name, you can either use the pre-defined COMPOSE_FILE environment variable, or use the `-f` option to specify the list of files.

Compose merges files in the order they're specified on the command line. Subsequent files may merge, override, or add to their predecessors.

For example:

```
$ docker compose -f compose.yaml -f compose.admin.yaml run backup_db
```

The `compose.yaml` file might specify a `webapp` service.

```
webapp:
  image: examples/web
  ports:
    - "8000:8000"
  volumes:
    - "/data"
```

The `compose.admin.yaml` may also specify this same service:

```
webapp:
  environment:
    - DEBUG=1
```

Any matching fields override the previous file. New values, add to the `webapp` service configuration:

```
webapp:
  image: examples/web
  ports:
    - "8000:8000"
  volumes:
    - "/data"
  environment:
    - DEBUG=1
```

Merging rules

- Paths are evaluated relative to the base file. When you use multiple Compose files, you must make sure all paths in the files are relative to the base Compose file (the first Compose file specified with `-f`). This is required because override files need not be valid Compose files. Override files can contain small fragments of configuration. Tracking which fragment of a service is relative to which path is difficult and confusing, so to keep paths easier to understand, all paths must be defined relative to the base file.

  > [!TIP]
  > You can use `docker compose config` to review your merged configuration and avoid path-related issues.

- Compose copies configurations from the original service over to the local one. If a configuration option is defined in both the original service and the local service, the local value replaces or extends the original value.

- For single-value options like `image`, `command` or `mem_limit`, the new value replaces the old value.
  original service:

```
services:
  myservice:
    # ...
    command: python app.py
```

  local service:

```
services:
  myservice:
    # ...
    command: python otherapp.py
```

  result:

```
services:
  myservice:
    # ...
    command: python otherapp.py
```

- For the multi-value options `ports`, `expose`, `external_links`, `dns`, `dns_search`, and `tmpfs`, Compose concatenates both sets of values:
  original service:

```
services:
  myservice:
    # ...
    expose:
      - "3000"
```

  local service:

```
services:
  myservice:
    # ...
    expose:
      - "4000"
      - "5000"
```

  result:

```
services:
  myservice:
    # ...
    expose:
      - "3000"
      - "4000"
      - "5000"
```

- In the case of `environment`, `labels`, `volumes`, and `devices`, Compose "merges" entries together with locally defined values taking precedence. For `environment` and `labels`, the environment variable or label name determines which value is used:
  original service:

  ```
  services:
    myservice:
      # ...
      environment:
        - FOO=original
        - BAR=original
  ```

  local service:

  ```
  services:
    myservice:
      # ...
      environment:
        - BAR=local
        - BAZ=local
  ```

  result:

  ```
  services:
    myservice:
      # ...
      environment:
        - FOO=original
        - BAR=local
        - BAZ=local
  ```

- Entries for `volumes` and `devices` are merged using the mount path in the container:
  original service:

```
services:
  myservice:
    # ...
    volumes:
      - ./original:/foo
      - ./original:/bar
```

local service:

```
services:
  myservice:
    # ...
    volumes:
      - ./local:/bar
      - ./local:/baz
```

result:

```
services:
  myservice:
    # ...
    volumes:
      - ./original:/foo
      - ./local:/bar
      - ./local:/baz
```

For more merging rules, see Merge and override in the Compose Specification.

Additional information

- Using `-f` is optional. If not provided, Compose searches the working directory and its parent directories for a `compose.yaml` and a `compose.override.yaml` file. You must supply at least the `compose.yaml` file. If both files exist on the same directory level, Compose combines them into a single configuration.
- You can use a `-f` with `-` (dash) as the filename to read the configuration from `stdin`. For example:

```
$ docker compose -f - <<EOF
  webapp:
    image: examples/web
```

```
    ports:
     - "8000:8000"
    volumes:
     - "/data"
    environment:
     - DEBUG=1
  EOF
```

When `stdin` is used, all paths in the configuration are relative to the current working directory.

- You can use the `-f` flag to specify a path to a Compose file that is not located in the current directory, either from the command line or by setting up a COMPOSE_FILE environment variable in your shell or in an environment file.

  For example, if you are running the Compose Rails sample, and have a `compose.yaml` file in a directory called `sandbox/rails`. You can use a command like docker compose pull to get the postgres image for the `db` service from anywhere by using the `-f` flag as follows: `docker compose -f ~/sandbox/rails/compose.yaml pull db`
  Here's the full example:

```
$ docker compose -f ~/sandbox/rails/compose.yaml pull db
Pulling db (postgres:18)...
18: Pulling from library/postgres
ef0380f84d05: Pull complete
50cf91dc1db8: Pull complete
d3add4cd115c: Pull complete
467830d8a616: Pull complete
089b9db7dc57: Pull complete
6fba0a36935c: Pull complete
81ef0e73c953: Pull complete
338a6c4894dc: Pull complete
15853f32f67c: Pull complete
044c83d92898: Pull complete
17301519f133: Pull complete
dcca70822752: Pull complete
cecf11b8ccf3: Pull complete
Digest: sha256:1364924c753d5ff7e2260cd34dc4ba05ebd40ee8193391220be0f9901d4e1651
Status: Downloaded newer image for postgres:18
```

Example

A common use case for multiple files is changing a development Compose app for a production-like environment (which may be production, staging or CI). To support these differences, you can split your Compose configuration into a few different files:

Start with a base file that defines the canonical configuration for the services.

`compose.yaml`

```
services:
  web:
    image: example/my_web_app:latest
    depends_on:
      - db
      - cache

  db:
    image: postgres:18

  cache:
    image: redis:latest
```

In this example the development configuration exposes some ports to the host, mounts our code as a volume, and builds the web image.

`compose.override.yaml`

```
services:
  web:
    build: .
    volumes:
      - '.:/code'
    ports:
      - 8883:80
    environment:
      DEBUG: 'true'

  db:
    command: '-d'
    ports:
```

```
      - 5432:5432

  cache:
    ports:
      - 6379:6379
```

When you run `docker compose up` it reads the overrides automatically.

To use this Compose app in a production environment, another override file is created, which might be stored in a different git repository or managed by a different team.

`compose.prod.yaml`

```
services:
  web:
    ports:
      - 80:80
    environment:
      PRODUCTION: 'true'

  cache:
    environment:
      TTL: '500'
```

To deploy with this production Compose file you can run

```
$ docker compose -f compose.yaml -f compose.prod.yaml up -d
```

This deploys all three services using the configuration in `compose.yaml` and `compose.prod.yaml` but not the dev configuration in `compose.override.yaml`.

For more information, see Using Compose in production.

Limitations

When merging Compose files, all relative paths (for build contexts, environment files, bind-mounted volumes, and other resources) are resolved relative to the base Compose file. The first file specified with `-f`, or `compose.yaml` in the current directory if `-f` is not used. Paths in override files are not resolved relative to the override file's own location.

This means that in a monorepo where Compose files are spread across team or component subdirectories, relative paths in those files are resolved incorrectly when used as override files.

> [!TIP]
>
> If you need each Compose file's paths to resolve relative to its own location, use the `include` top-level element instead of merging with `-f`. Each included file is loaded with its own project directory, so relative paths resolve correctly.

Reference information

- Merge rules

□□□ #2
□ □□□ □□□ 2026-10-31 19:08:32 UTC
□ □□□ □□□ 2026-10-31 19:08:34 UTC